

Gestion des transactions et de la concurrence

Chapitre 3

Hmida HMIDA

Février 2023

DSI2 - ISET Bizerte

PLAN

1. Introduction
2. Problèmes de la concurrence
3. Contrôle de la concurrence
4. Algorithmes de contrôle de la concurrence
5. Les transactions dans Oracle

Introduction

Une base de données est une ressource :

- Partagée entre plusieurs utilisateurs (Multi-utilisateurs)
- Traitée de manière concurrente (Multi-programmation)
 - Traitement entrelacé

Les problèmes possibles :

- Incohérence de données
- Interblocage
- Pannes (disque, système, exception)

DÉFINITION

Transaction

Une opération ou série d'opérations effectuées par un seul utilisateur ou programme pour lire ou mettre à jour le contenu de la base de données. Elle permet de passer d'un état cohérent de la base de données à un autre.

- 2 opérations considérées : Lire(), Écrire()
- Une séquence est terminée par validation Commit ou une annulation Rollback de façon implicite ou explicite
- Toute opération d'un programme est exécutée dans une transaction
- Un programme est décomposé en 1 ou plusieurs transactions
- Des exécutions différentes du même programme produit des transactions différentes

OPÉRATIONS DE L/E

- Lire(X)
 - Lecture d'un objet X de la BD dans une variable X
 - Trouver son adresse dans le bloc disque et le charge en mémoire
- Écrire(X)
 - Écriture de la valeur d'une variable X dans un objet X de la BD
 - Recherche du bloc, copie en mémoire, sauvegarde dans le disque du bloc mis à jour
- Transaction de lecture-seulement (Read Only)
- Transaction de lecture-écriture (Read Write)
- Ensembles de lecture/écriture : les objets de la BD accédés en L/E

EXEMPLE

Transaction T1

```
Lire(X);  
X:=X+N;  
Écrire(X);  
Lire(Y);  
Y:=Y-N;  
Écrire(Y);
```

Transaction T2

```
Lire(X);  
X:=X+M;
```

- **T1** est de type lecture-écriture
- **T2** est de type lecture seulement
- l'ensemble de lecture de **T1** est $\{X,Y\}$
- l'ensemble de lecture de **T2** est $\{X\}$
- l'ensemble d'écriture de **T1** est $\{X,Y\}$

TRANSACTION ACID

- **Atomicité** : une transaction est exécutée soit en totalité soit pas du tout
 - Une transaction qui ne peut pas se terminer doit être annulée par le SGBD (Rollback)
- **Cohérence** : une transaction respecte les contraintes d'intégrité de la BD et préserve sa cohérence ou consistance.
 - Une transaction mène la BD d'un état cohérent vers un autre état cohérent
 - Tâche assurée par le programmeur et le module du SGBD activant les contraintes d'intégrité
- **Isolation**: une transaction ne voit pas les effets des autres transactions qui s'exécutent en même temps
 - Les écritures des autres transactions en cours sur les données manipulées par la transaction en question ne sont pas visibles
 - Conséquence: l'exécution de plusieurs transactions concurrentes est équivalente à une exécution séparée, en série des transactions
- **Durabilité**: les effets d'une transaction validée sont persistants et ne sont jamais perdus

Problèmes de la concurrence

PERTE DE MISE À JOUR

Exécution entrelacée de 2 transactions accédant au même objet.

Temps	T1 (Agence 1)	T2 (Agence 2)	Solde
t_0			500
t_1	Lire(Solde);		500
t_2	Solde := Solde -200;		500
t_3		Lire(Solde);	500
t_4		Solde := Solde + 90;	500
t_5	Écrire(Solde);		300
t_6		Écrire(Solde);	590

Solde a une valeur incorrecte (590 au lieu de 390) parce que la mise à jour par T1 est perdue.

ANALYSE INCOHÉRENTE

Une transaction lit plusieurs valeurs pour faire une somme, par exemple, et une autre modifie une ou plusieurs de ces valeurs.

Temps	T1 (Agence 1)	T2 (Agence 2)	X	Y	Z
t_0			500	200	
t_1		Z:=0;	500	200	0
t_2	Lire(X);		500	200	0
t_3	X:=X-100;		500	200	0
t_4	Écrire(X);		400	200	0
t_5		Lire(X);	400	200	0
t_6		Z := Z + X;	400	200	400
t_7		Lire(Y);	400	200	400
t_8		Z:=Z + Y;	400	200	600
t_9	Lire(Y);		400	200	600
t_{10}	Y:=Y+100;		400	200	600
t_{11}	Écrire(Y);		400	300	600

Valeur incorrecte de **Z** car Y est modifiée par T1 après sa lecture par T2.

DÉPENDANCES NON VALIDÉES

Lecture sale (Dirty Read) : lire une valeur modifiée par une autre transaction mais non validée.

Temps	T1 (Agence 1)	T2 (Agence 2)	Solde
t_0			500
t_1	Lire(Solde);		500
t_2	Solde := Solde - 100;		500
t_3	Écrire(Solde);		400
t_4		Lire(Solde);	400
t_5		Solde := Solde - 200 ;	400
t_6	Rollback;		500
t_7		Écrire(Solde);	200

Solde a une valeur incorrecte (200 au lieu de 300) parce que la mise à jour par T1 est annulée par Rollback après sa lecture par T2.

Écriture sale : lire une valeur modifiée par une autre transaction mais non validée.

LECTURES NON REPRODUCTIBLES

Une transaction effectue 2 lecture du même objet et obtient des valeurs différentes

Temps	T1 (Agence 1)	T2 (Agence 2)	Solde
t_0			500
t_1	Lire(Solde);		500
t_2	Int := Solde * 0.01;		500
t_3		Lire(Solde);	500
t_4		Solde := Solde - 200 ;	500
t_5		Écrire(Solde);	300
t_6	Lire(Solde);		300
t_7	Solde:=Solde + Int;		300
t_8	Écrire(Solde);		305

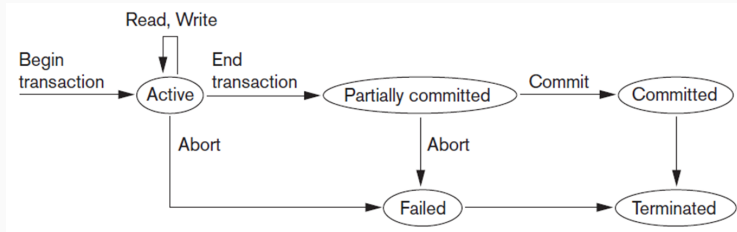
Dans T1, la première lecture donne la valeur 500 et la seconde la valeur 300.

Contrôle de la concurrence

OBJECTIFS

- Produire une exécution correcte, en respectant les propriétés ACID
- Assurer l'**Isolation** des transactions tout en
 - Permettant l'exécution simultanée d'un grand nombre de transactions
 - Gardant de très bonne performance
 - Évitant les blocages
- Les autres propriétés sont assurées par la reprise après les pannes
- L'exécution doit être équivalente à une **exécution en série** des transactions
 - Garantie l'isolation des transactions
 - **Exécution** sérialisable = exécution équivalente à une exécution en série quelconque des transactions
 - Plusieurs exécutions en série sont possibles ($n!$ pour n transactions)
- Objectif contrôle de concurrence = produire des exécutions sérialisables en modifiant l'entrelacement (ordonnancement) des transactions

DIAGRAMME D'ÉTATS-TRANSITION D'UNE TRANSACTION



Le SGBD doit mémoriser les phases d'exécution d'une transaction :

- BEGIN_TRANSACTION
- READ ou WRITE
- END_TRANSACTION
- COMMIT_TRANSACTION
- ROLLBACK (ou ABORT)

OPÉRATIONS COMMUTABLES/CONFLICTUELLES

Opérations conflictuelles

Deux opérations a et b , exécutées respectivement par deux transactions différentes T et T' sont dites conflictuelles si l'exécution des deux séquences $a;b$ (a puis b) et $b;a$ (b puis a) est **susceptible** de conduire à des résultats différents et ne sont pas commutables.

- Les opérations d'une même transaction ne peuvent pas changer de position relative
- Deux opérations sur des articles différents sont toujours commutables
- Seules les lectures (Lire-Lire) sont commutables \rightarrow s'il y a au moins une écriture alors il y a conflit.

Conflit équivalence

Deux exécutions sont conflit équivalentes s'ils ont le même ordre relatif de chaque paire d'opérations conflictuelles.

NOTATIONS

- Transaction T_i : Séquence d'opérations terminée par Commit/Rollback.
- Opérations: Lire, Écrire, Commit, Rollback
 - $l_i[x]$: lecture de l'article x dans la transaction T_i
 - $e_i[x]$: écriture de l'article x dans la transaction T_i
 - c_i : validation (commit) de la transaction T_i
 - r_i : annulation (rollback) de la transaction T_i
- Ordonnancement (H_i): ordre d'exécutions des opérations atomiques (L/E) des différentes transactions
- Exemple :
 - $T_1 : l_1[x]e_1[x]c_1; T_2 : l_2[y]e_2[y]c_2$
 - $H_1 : l_1[x]e_1[x]c_1l_2[y]e_2[y]c_2$ (une exécution série de T_1 puis T_2)
 - $H_2 : l_1[x]l_2[y]e_1[x]e_2[y]c_2c_1$ (une exécution concurrente de T_1 et T_2 équivalent à H_1)

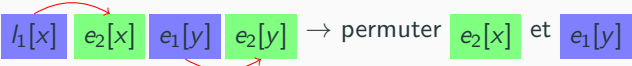
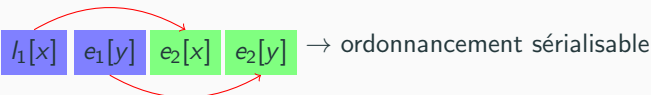
SÉRIALISABILITÉ

Sérialisabilité

Un ordonnancement de n transactions est sérialisable ou conflit sérialisable s'il est conflit équivalent à un ordonnancement série des mêmes transactions.

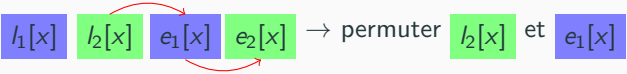
- Donne une exécution correcte (Cohérence)
- Vérification de sérialisabilité :
 - Transformations par commutation des opérations non conflictuelles.
 - Le graphe de précédence est acyclique.

Exemple 1 de transformations par commutation :

- H_1 :  \rightarrow permuter $e_2[x]$ et $e_1[y]$
- H_1 :  \rightarrow ordonnancement sérialisable

SÉRIALISABILITÉ

Exemple 2 de transformations par commutation :

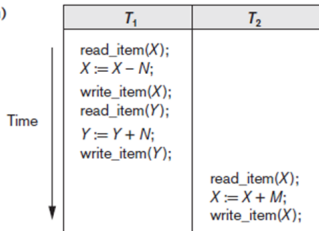
- H_1 :  \rightarrow permuter $l_2[x]$ et $e_1[x]$ est impossible (opérations conflictuelles)
- H_1 ordonnancement non sérialisable

Graphe de précédence ou graphe de sérialisation :

- Nœuds : Les transaction T_i
- Arcs : Pour toute paire d'opérations conflictuelle A_i précède B_j un arc orienté de T_i vers T_j
- Si pas de cycle alors c'est une exécution sérialisable

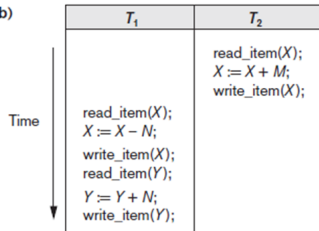
GRAPHE DE PRÉCÉDENCE

(a)



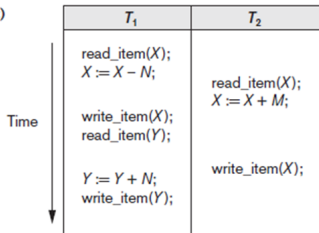
Schedule A

(b)

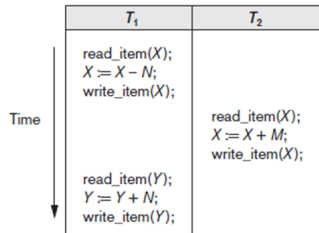


Schedule B

(c)

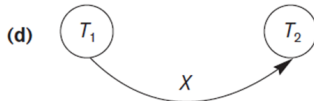
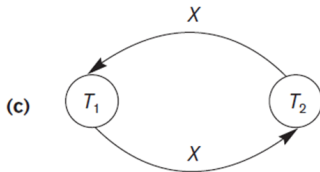
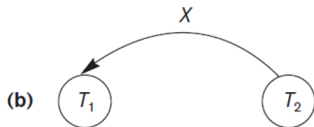
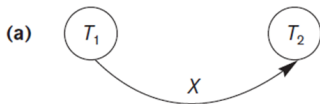


Schedule C



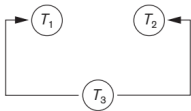
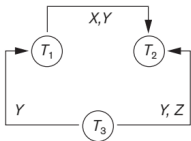
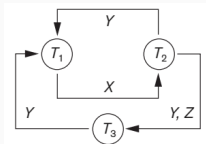
Schedule D

GRAPHE DE PRÉCÉDENCE



- (a) et (d) sérialisables et équivalents à $T_1; T_2$
- (b) sérialisable et équivalent à $T_2; T_1$
- (c) non sérialisable

GRAPHE DE PRÉCÉDENCE



Non sérialisable : 2 cycles

- $T_1 \rightarrow T_2 \rightarrow T_1$
- $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$

Sérialisable et équivalent à

- $T_3 \rightarrow T_1 \rightarrow T_2$

Sérialisable et équivalent à

- $T_3 \rightarrow T_1 \rightarrow T_2$
- $T_3 \rightarrow T_2 \rightarrow T_1$

Algorithmes de contrôle de la concurrence

APPROCHES DE CONTRÔLE DE CONCURRENCES

- Le théorème de sérialisabilité est un moyen de vérification : il ne permet pas de générer des exécutions sérialisables
- Ordonnanceur (Scheduler):
 - Un module du SGBD qui applique le protocole de contrôle de concurrence
 - Réordonne les opération pour garantir la sérialisaibilité
- Deux familles d'approches pour garantir la sérialisibilité :
 - Optimistes :
 - Supposent que le concurrence est rare
 - Agissent suite à la détection d'un cycle pour rejeter une transaction
 - Estampillage
 - Pessimistes :
 - Préviennent l'apparition de cycles en évitant l'accès concurrent
 - Verrouillage

VERROUILLAGE

Verrou (Lock)

- Une variable associée à chaque objet de la BD (enregistrement)
- Objectif : bloquer les autres transactions qui veulent accéder à un enregistrement en cours de mise à jour par une transaction

Principe : Chaque transaction qui va modifier une donnée X

- Vérifie si X n'est pas verrouillée (sinon attendre)
- Poser un verrou
- Modifier X
- Enlever le verrou : libérer X \rightarrow servir une transaction bloquée

Verrouillage à 2 phase (2PL) : 2 Phase Lock

- L'algorithme le plus ancien et le plus utilisé

VERROUILLAGE À 2 PHASES

Deux types de verrou :

- Verrou partagé **S** (Shared Lock) : il est possible de poser d'autres verrou de type partagé (pour lecture)
- Verrou exclusif **X** (Exclusive Lock) : il est impossible de poser un autre verrou partagé ou exclusif (lecture et écriture)

Règle des deux phases

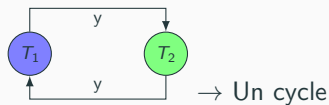
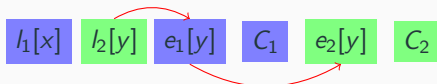
Une transaction qui libère un verrou n'a plus le droit d'en poser.

Deux Phases :

1. Poser des verrous
2. Libérer des verrous (à la fin de la transaction)

EXEMPLE DE VERROUILLAGE

Prenons l'exécution concurrente :

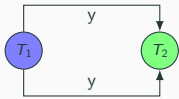
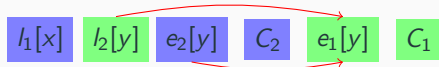


Par application du verrouillage :

1. $l_1[x]$: verrou S sur x donné à T_1
2. $l_2[y]$: verrou S sur y donné à T_2
3. $e_1[y]$: T_1 est bloquée (demande un verrou X sur y verrouillé par T_2 avec S)
4. C_1 : T_1 est bloquée
5. $e_2[y]$: verrou X sur y donné à T_2 (Upgrade du verrou S)
6. C_2 : validation de T_2 et libération de ses verrous
7. Reprise de T_1 : exécution de $e_1[y]$ et C_1

EXEMPLE DE VERROUILLAGE

Exécution obtenue :



→ Pas de cycle

PROBLÈMES DU VERROUILLAGE

1. Interblocage (Deadlock):

- T_1 a un verrou sur x et veut accéder à y qui est verrouillé par T_2 et T_2 veut accéder à x .
- Solutions :
 - Timeout : Au bout d'un temps d'attente la transaction bloquée est annulée
 - Autres : Wait-Die, Wound-Wait, Victim Selection, ...

2. Performance :

- Verrouillage d'enregistrement coûteux
- Verrouillage de table = temps d'attente important

3. Famine (Starvation) :

- Une transaction attend indéfiniment et les autres continuent leurs exécutions
- Solution : FIFO

Estampille (Timestamp)

- Un numéro distinct attribué à chaque transaction croissant avec la date de début noté $E(T_i)$
- Donnée par un compteur ou horloge
- Mémoriser les estampilles des transactions les plus jeunes effectuant une lecture/écriture notées resp. $E_l(x)$ et $E_e(x)$

Principe :

- Les opérations en conflits passent selon l'ordre des estampilles
- En cas de problème annuler la transaction la plus récente
- Pas de verrouillage (et donc d'interblocage)

ESTAMPILLAGE DE BASE

1. Lecture : $r_i[x]$

- si $E(T_i) \geq E_e(x)$ alors lecture autorisée et mise à jour ($E_l(x) = \max(E_l(x), E(T_i))$)
- sinon T_i est annulée et redémarrée avec une nouvelle estampille. T_i essaye de lire un enregistrement qui a été modifié par une transaction plus récente

2. Écriture : $e_i[x]$

- si $E(T_i) < E_l(x)$: T_i essaye d'écrire en retard un enregistrement qui a été lu par une transaction plus récente. T_i est alors annulée
- si $E(T_i) < E_e(x)$: T_i essaye d'écrire une valeur obsolète d'un enregistrement qui a été modifié par une transaction plus récente. T_i est alors annulée
- sinon ($E(T) \geq E_l(x)$ et $E(T_i) \geq E_e(x)$) l'écriture est autorisée et mise à jour ($E_e(y) = E(T_i)$)

Problèmes : Annulations en cascade + famine pour les longues transactions.

EXEMPLE ESTAMPILLAGE

- $H_1 : l_1[x] \ e_2[x] \ l_3[x] \ l_2[x] \ e_1[x]$
- Estampille : indice de la transaction
- Exécution :
 1. $l_1[x]$: acceptée
 2. $e_2[x]$: en conflit avec $l_1[x]$ mais estampille plus grande \rightarrow acceptée
 3. $l_3[x]$: en conflit avec $e_2[x]$ mais estampille plus grande \rightarrow acceptée
 4. $l_2[x]$: pas de conflit \rightarrow acceptée
 5. $e_1[x]$: en conflit avec $e_2[x]$ et estampille plus petite \rightarrow rejetée et redémarrage de la T_1 avec estampille
 6. $l_1[x] \ e_2[x] \ l_3[x] \ l_2[x] \ e_1[x] \ R_1 \ l_4[x] \ e_4[x]$

Les transactions dans Oracle

VALIDATION ET ANNULATION

- Une transaction commence avec la première instruction et se termine par :
 - Ordre ROLLBACK ou COMMIT
 - Déconnexion (valide de la transaction en cours)
 - Erreur système ou programme (annule la transaction en cours)
 - Validation automatique après un ordre DDL
- Validation COMMIT
 - COMMIT [WORK] [COMMENT 'Commentaire'] [WRITE WAIT|NOWAIT IMMEDIATE | BATCH] [FORCE clause];
 - WAIT | NOWAIT : synchronisation par rapport au Redo Log (par défaut WAIT).
 - IMMEDIATE | BATCH : écriture du Redo Log (par défaut IMMEDIATE).
- Point de sauvegarde SAVEPOINT
 - SAVEPOINT savepoint_name;
- Annulation ROLLBACK
 - ROLLBACK [WORK] [TO [SAVEPOINT] savepoint_name | FORCE 'commentaire'];
 - Annule les commandes depuis le dernier commit ou depuis le point de sauvegarde indiqué.

NIVEAUX D'ISOLATION DANS SQL

Objectif : Augmenter la concurrence en relaxant les contraintes d'isolation

- La sérialisabilité est coûteuse
- L'isolation assure la cohérence de toutes les transactions mais certaines transactions tolèrent un niveau inférieur

SQL définit 4 niveaux par la commande

SET TRANSACTION ISOLATION LEVEL **SERIALIZABLE** | **REPEATED READ** | **READ COMMITTED** | **READ UNCOMMITTED**

Niveau (Du plus souple au plus strict)	Lectures sales	Lectures non répétées	Lectures fantômes¹
READ UNCOMMITTED	X	X	X
READ COMMITTED (l'écriture bloque la lecture)		X	X
REPEATABLE READ (lecture bloque aussi l'écriture)			X
SERIALIZABLE			

¹ Un enregistrement inséré par une autre transaction qui répond aux critères des enregistrements de la transaction en cours (comme analyse incohérente)

NIVEAUX D'ISOLATION ORACLE

Deux niveaux d'isolation + possibilité de restreindre les écritures

Commande :

```
SET TRANSACTION [ READ ONLY | READ WRITE ]  
                [ ISOLATION LEVEL [ SERIALIZE | READ COMMITTED ] ]  
                [ USE ROLLBACK SEGMENT 'segment_name' ]  
                [ NAME 'transaction_name' ];
```

- READ ONLY | READ WRITE : empêcher les écritures ou non
- ISOLATION LEVEL SERIALIZABLE | READ COMMITTED : choisir le niveau d'isolation.
- USE ROLLBACK SEGMENT 'segment_name' : choisir le segment utilisé pour la récupération
- NAME 'transaction_name' : Donner un nom à la transaction

Exemple : SET TRANSACTION READ ONLY ISOLATION LEVEL READ COMMITTED;

VERROUILLAGE DANS ORACLE

Principe hiérarchique :

- Verrouillage multi-granularité : db, table, row, ...
- Verrouillage descendant: implicite
 - Un verrou à un niveau est implicitement valable pour toutes les données de niveau inférieur incluses
 - Ex. Un verrou en écriture sur une table est valable pour tous les articles de la table et pour tous les champs de ces articles
- Verrouillage ascendant: verrous d'intention
 - Pour réaliser une opération à un niveau, il faut aussi vérifier s'il n'y a pas conflit avec les verrous des niveaux supérieurs
 - Ex. Lorsqu'on veut écrire dans un article, il faut voir s'il n'y a pas des verrous en lecture/écriture sur la table
 - Méthode: utilisation de verrous supplémentaires dits d'intention
 - Règle: avant de réaliser une opération à un niveau, il faut demander le verrou d'intention pour l'opération au niveau supérieur (voir tableau compatibilité)

VERROUILLAGE DANS ORACLE

- 5 types :
 - S (Shared, partagé, en lecture)
 - X (eXclusive, exclusif, en écriture)
 - IS (Intention Shared) : Verrou S sur un descendant
 - IX (Intention eXclusive) : Verrou X sur un descendant
 - SIX (Shared Intention eXclusive) : nœud courant verrouillé S et intention de verrou X sur descendants
- Matrice de compatibilité

	IS	IX	S	SIX	X
IS	true	true	true	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false

- Pour poser un verrou S ou IS, il faut avoir le verrou IS ou IX sur le parent
- Pour poser une verrou X, IX ou SIX, il faut avoir le verrou IX ou SIX sur le parent

VERROUILLAGE DANS ORACLE

- verrous dans Oracle :
 - Ligne : X (écriture)
 - Table : IS, IX, S, SIX, X
- Verrouillage implicite
 - SELECT ... FROM ... WHERE: aucun verrou
 - INSERT/UPDATE/DELETE: X (ligne), IX (table)
 - SELECT ... FOR UPDATE: X (ligne), IS (table)
- Verrouillage explicite : seulement au niveau des tables
 - Commande LOCK TABLE :
LOCK TABLE tables IN type MODE;
lock_mode peut avoir les valeurs suivantes :
 - type: ROW SHARE (IS), ROW EXCLUSIVE (IX), SHARE (S),
SHARE ROW EXCLUSIVE (SIX), EXCLUSIVE (X)

Questions?